

Sissejuhatus infotehnoloogiasse

Tarkvara arhitektuur

- Arhitektuurinduse ja arendustöö põhimõtteid
- Komponendid: terviksüsteemid, serverid, frameworkid, teegid
- Arhitektuurinäiteid
- Programmeerimiskeelte tüübid eri vaadetest
- Domineerivad programmeerimiskeeled
- Arendusvahendid
- Tarkvara & tootmine: ajaloolised faasid
- Võrgunduse areng
- Süsteemide integratsioon
- Vabatarkvara

Tarkvara arhitektuur

- Arhitektuuri all mõeldakse IT-s:
 - mingi süsteemi tehnoloogilisi põhimõtteid
 - millisteks suurteks osadeks süsteem jaotub
 - kuidas osad omavahel suhtlevad
 - milliseid suuri valmistükke süsteem kasutab

- Tarkvaras on sellisteks küsimusteks näiteks:
 - Mis opsüsteemi alla rakendus teha
 - Kas hoida infot lihtsalt failides või andmebaasis
 - Millist veebiserverit ja kuidas täpselt kasutada
 - Kas pöörduda andmebaasi poole otse või csv või XML või json vahekihi abil
 - Mis keeles/keeltes rakendus teha
 - Millist kasutajaliidese teeki kasutada
 - ... jne

Arendaja analoog on arhitekt/ehitusinsener

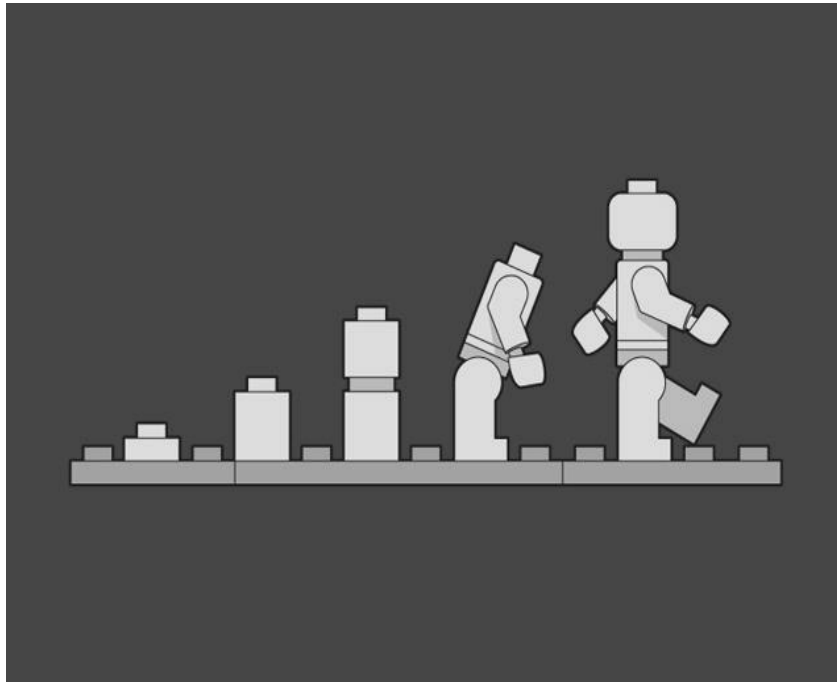
Tarkvara lihtne analoogia ehitusega ei ole eriti pädev:

- Tarkvara on kõige keerulisem asi maailmas: tarkvarasüsteemidel on kõige rohkem osi
- Arendaja kirjutatud programm ongi tegelikult ehitusjoonis, mitte ehitus: ehitus (töötav tarkvara) tehakse kompilaatorite, linkurite jms abil programmist automaatselt.
- Programmeerija “analoog” on arhitekt/ehitusinsener.
- Tarkvara arenduses ei ole nõ lihtsaid “ehitusmehi” vaja.
- Täiendava tööjõu lisamine projekti käigus teeb projekti täitmise reeglina veel aeglasemaks.

Tarkvara ei ole lego

Tarkvara ei panda kokku “nagu lego blokkidest”:

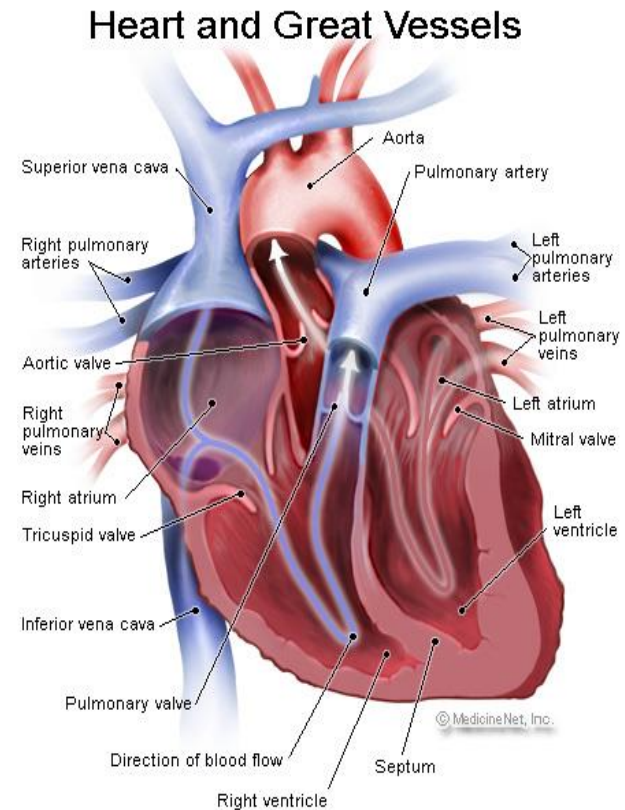
- Lego blokke saab ühendada väga lihtsalt, ja neil ei ole sisemist keerulist ehitust



Tarkvara on sarnane bioloogiaga

Tarkvara ei panda kokku “nagu lego blokkidest”:
ta sarnaneb enim **bioloogiaga**

- “Tarkvarablokid” on väga keerulised süsteemid, ja nende ühendamine tähendab nõ tavapärast programmeerimist
- Tarkvara areneb olemasolevast tarkvarast arendajate abil evolutsiooni teel, sarnaselt bioloogiaga



Arhitektuuri eesmärgid: lihtsus mitmekesisuses

Arendamise ja haldamise lihtsus on kõige tähtsam.

- Valiku juures on otsustav see, mida arendajad/haldajad kõige paremini oskavad kasutada.
- Lisatükkide kasutamist ilma selge vajaduseta tuleks vältida.
- Abstraktsioonid tilguvad läbi.
- Ei ole olemas iga juhu jaoks sobilikke tehnoloogiaid/komponente

No silver bullet!



Tarkvarasüsteemide komponendid

- **Tarkvarasüsteemid** ehitatakse reeglina mitmesuguste komponentide kokkupaneku, s.t. kokkuprogrammeerimise teel, või teisiti öeldes: komponente kasutades.

Neid komponente võib klassifitseerida - näiteks - järgmisel viisil:

- Terviklikud lõppkasutaja-rakendusprogrammid
- Suured “valmiskomponendid”, näiteks andmebaasimootorid
- Raamistikud ehk frameworks
- Teegid ehk libraries

Tarkvarasüsteemide komponendid

■ Terviklikud lõppkasutaja-rakendusprogrammid,

Neid saab tihti juhtida ja mõne teise tarkvarapaketiga programmiselt siduda, a la:

- Tekstitöötlus, näiteks Word või LibreOffice
- Tabelarvutus, näiteks Excel
- Raamatupidamise tarkvara
- Arendustarkvara, näiteks Eclipse või vscode
- Veebibrauser

Tarkvarasüsteemide komponendid

■ Suured valmiskomponendid

Andmebaasiserverid, www-serverid, mailiserverid, graafikaserverid nagu X11, operatsioonisüsteem ise jne

Need programmid on tehtud eeskätt selleks, et lõppkasutaja jaoks loodud rakenduste tegemist hõlbustada. Näitena andmebaasi väljakutsumine Pythonist:

```
.....
cursor=con.cursor()
sqlstr= """  select clientid, clientname
              from clientbase
              where clientname like '%Jaan%' """
cur.execute(sqlstr)
results=cur.fetchall()
for i in results
    print "id: ",i[0], " name: ",i[1]
.....
```

Tarkvarasüsteemide komponendid

■ Raamistikud ehk frameworks

Edasiarendamiseks ja ümbertegemiseks mõeldud terviklikud näiterakendused, levinud eeskätt „tüüpiliste“ andmebaasi-kesksete veebirakenduste jaoks. Igast hästi läbimõeldud rakendusest võib saada selline näiterakendus. Populaarseid raamistikke:

- Microsoft .NET
- Java Spring
- Ruby on Rails
- PHP Zend framework
- Python Django
- Javascript Angular

Raamistik ei lase arendajal vabalt valida, kuidas süsteem peaks töötama, vaid pigem suunab teda täiendama etteantud näitesüsteemi, mille raamistiku arendaja on välja töötanud.

Tarkvarasüsteemide komponendid

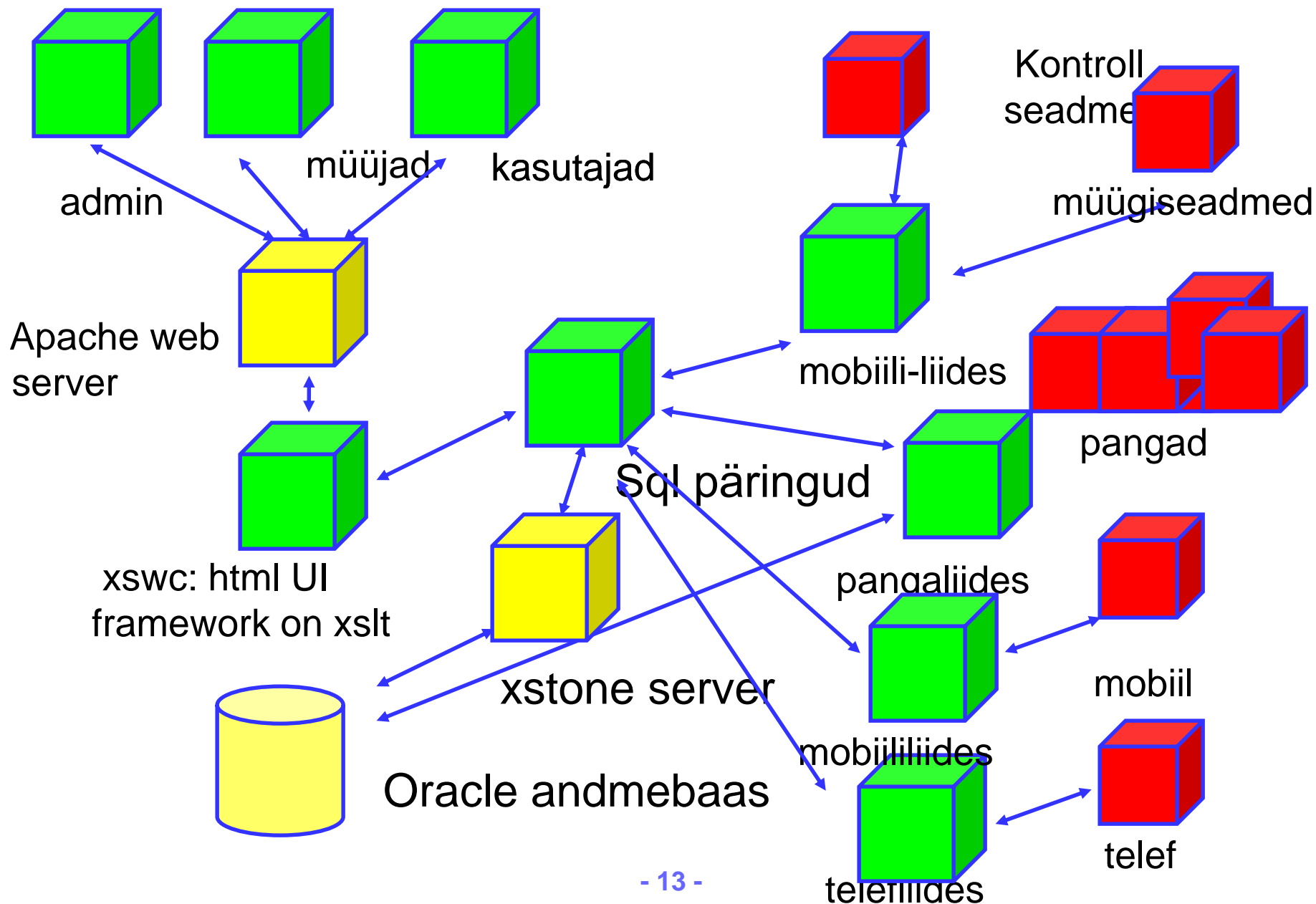
■ Teegid ehk libraries

Konkreetseid, piiratud funktsioone realiseerivad väikesed komponendid ja nende komplektid. Mõned komplektid on hiigelsuured. Funktsioonide näiteid:

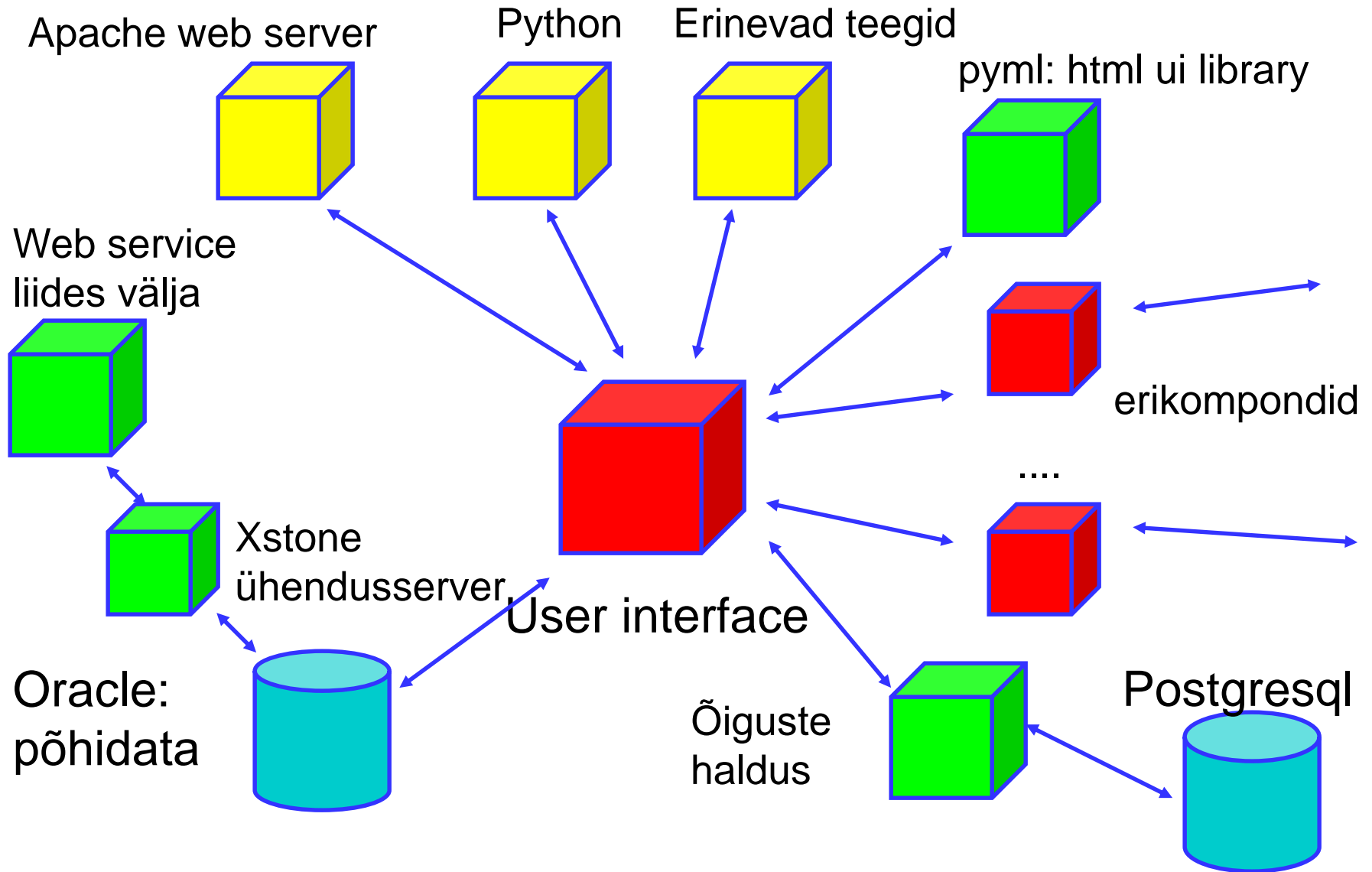
- Trükkimine
- Faili kirjutamine
- Võrguühenduse avamine
- Ringi, joone vms asja joonistamine

Selliseid komponente levitatakse enamasti komplekti kompilaatorite ja muude tarkvara-arendusvahenditega ning nad on tüüpiliselt kasutatavad ainult selle konkreetse programmeerimiskeele ja arendusvahendi koosseisus.

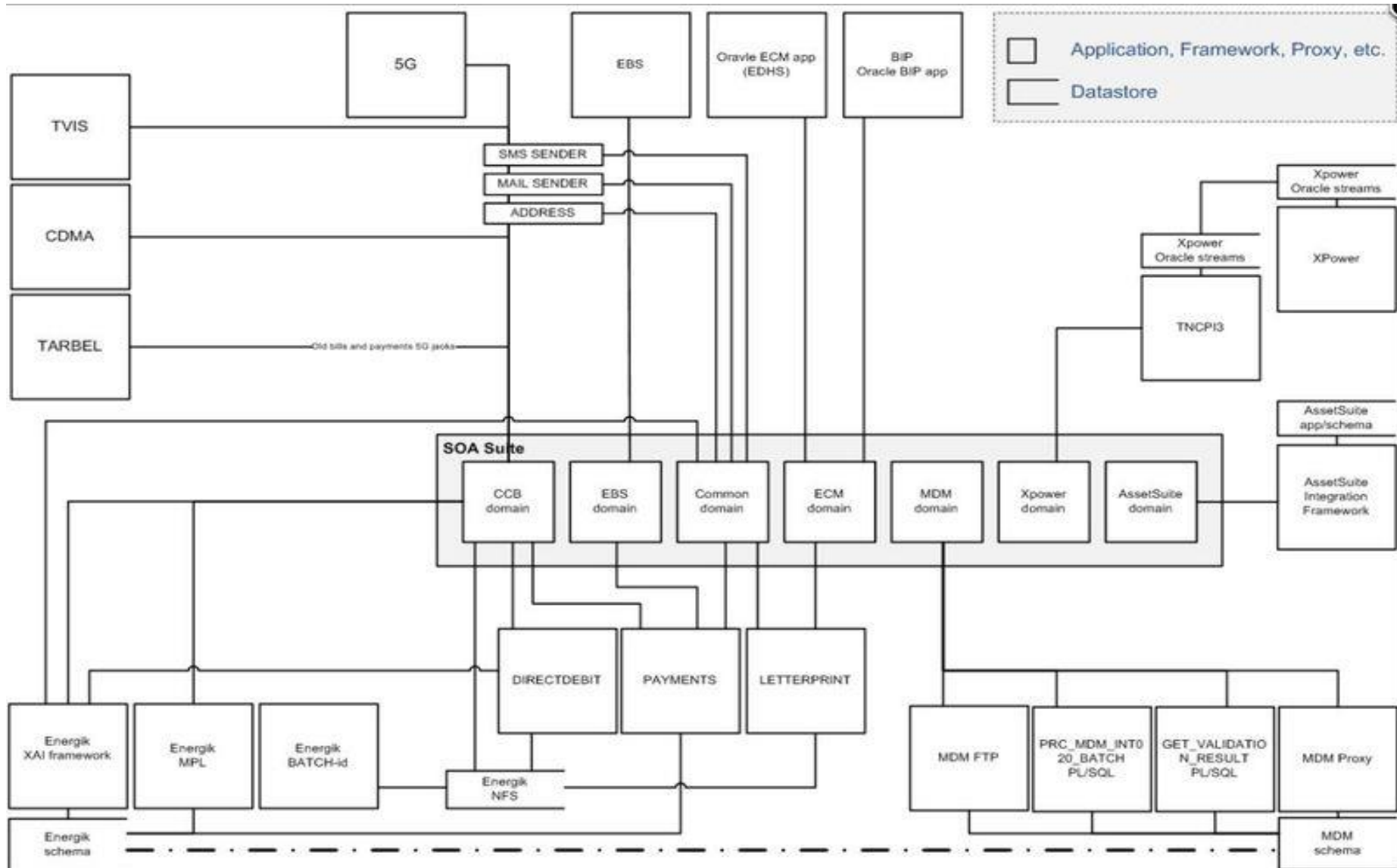
Arhitektuur: piletisüsteemi näide



Riikliku registri näide

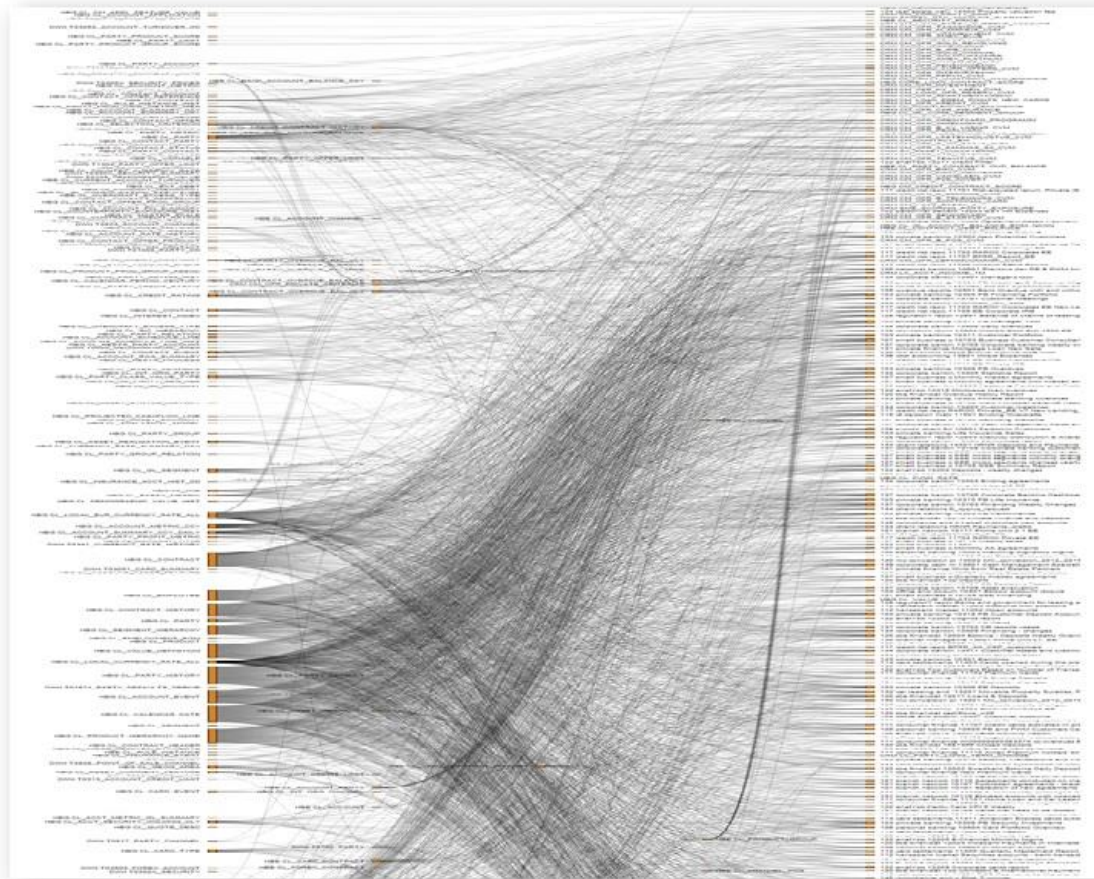


Üks suur eesti infosüsteem: iga blokk on omaette keerukas süsteem oma andmebaasidega



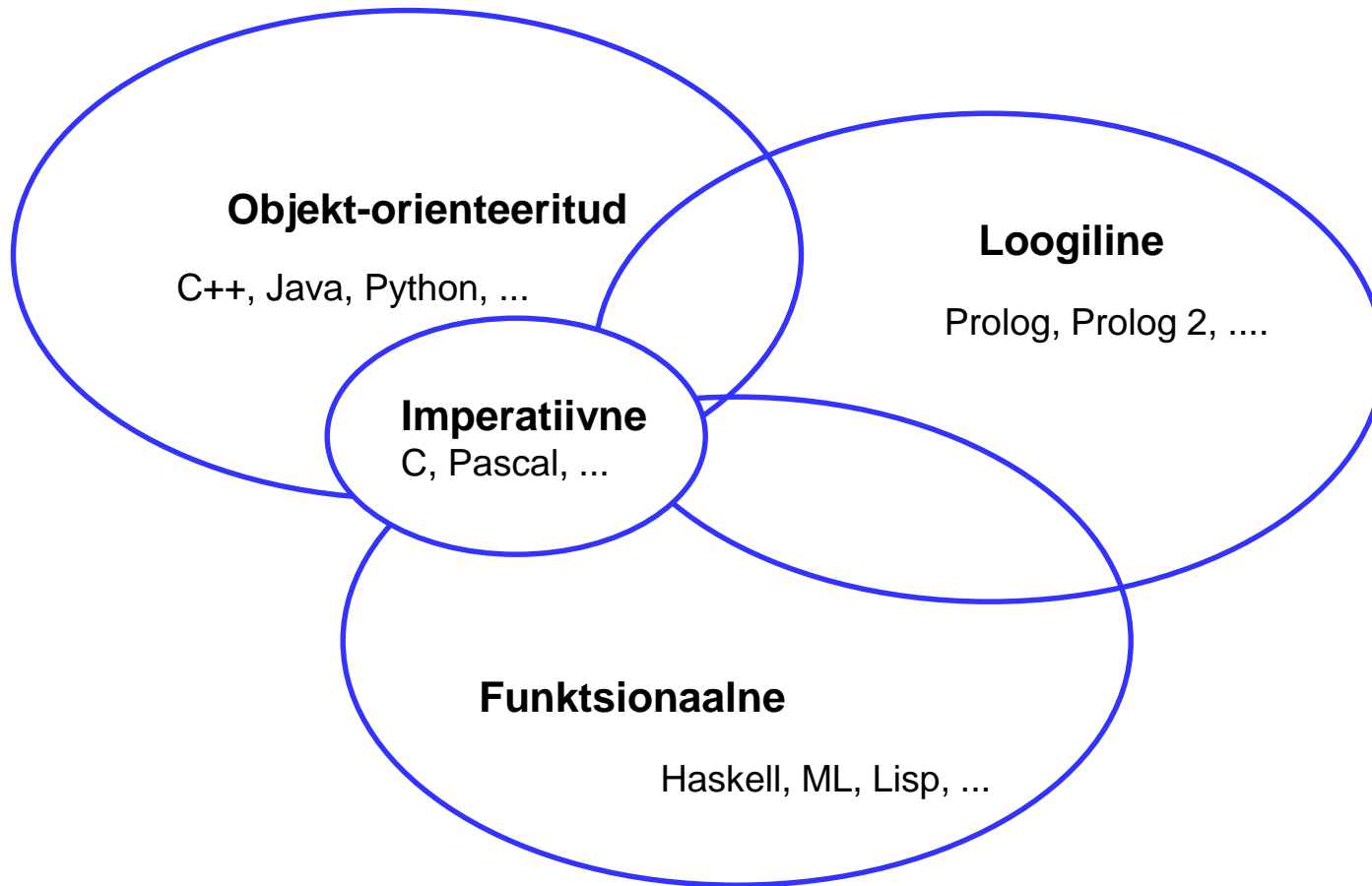
Andmete liikumise keerulised ahelad

- Vaata dlineage.com



MeeldProgrammeerimiskeelte paradigmad

Vaade 1: suuremad algoritmikeelte paradigmad erivahendite järgi, mida keel pakub:



Meeldetuletus:

algoritmilised keeled ja kirjelduskeeled

- **Algoritmilised (programeerimiskeeled)**
 - C, C++, Basic, Java, Python, PHP, assembler
- **Kirjelduskeeled (spetsifitseerimiskeeled)**
 - HTML, SQL, XML, RDF, TeX,...
- **Teoorias korruga mõlemat: loogika**
- **Praktiline süntees:**
 - algoritmilised keeled manipuleerivad kirjelduskeele abil antavate objektidega
 - näited: Javascript ja HTML., C/Java/Python/... ja SQL,

- Enamik keeli sisaldab suhteliselt piiratud võimalustega standardteeke.

Näide: C standardteek sisaldab stringitöötlust, failitöötlust, trükkimist, veel paari analoogilist kategooriat.

- Reeglina on võimsad rakendusteedid (graafika, aknad, hiir, võrguühendused, paralleelprotsessid jne ...) ebastandardised, piiratud teatud riistvara ja opsüsteemiga (Windows, Linux, iOS, Android, ...)
- Erandina on Javal väga mahukas standardteek, mis samas porditud kõigile olulistele opsüsteemidele.

Valdkonniti domineerivad programmeerimiskeeled

- Universaalseim, välja arvatud tippkiirust nõudvad või embedded või brauseri- või operatsioonisüsteemi-rakendused: **Java**
- Maksimaalset kiirust nõudvad rakendused, embedded ja süsteemprogrammeerimine: **C, C++**
- Andmetöötlus ja skriptid ilma kasutajaliideseta: **Python, Java, C, Go, Perl, Ruby**
- Windowsi kasutajaliidesega rakenduste programmeerimine: **C#, VisualBasic, C, (Java)**
- Maci ja iPhone programmeerimine: **Swift, Objective-C**
- Androidi programmeerimine: **Java, (C)**
- Veebibrauseri programmeerimine: **Javascript**
- Veebirakenduse programmeerimine: **PHP, Javascript, Python, Ruby, Java, Go, C , Perl ..**
- Spetsiifilised rakendused: vastavalt vajadusele

Programmi töötamiseks:

- Kompilaator
- Interpretaator

Programmi kirjutamiseks:

- Sobivad tekstiredaktorid
- Visuaalsed arendusvahendid

Suure hulga lähtekoodi halduseks:

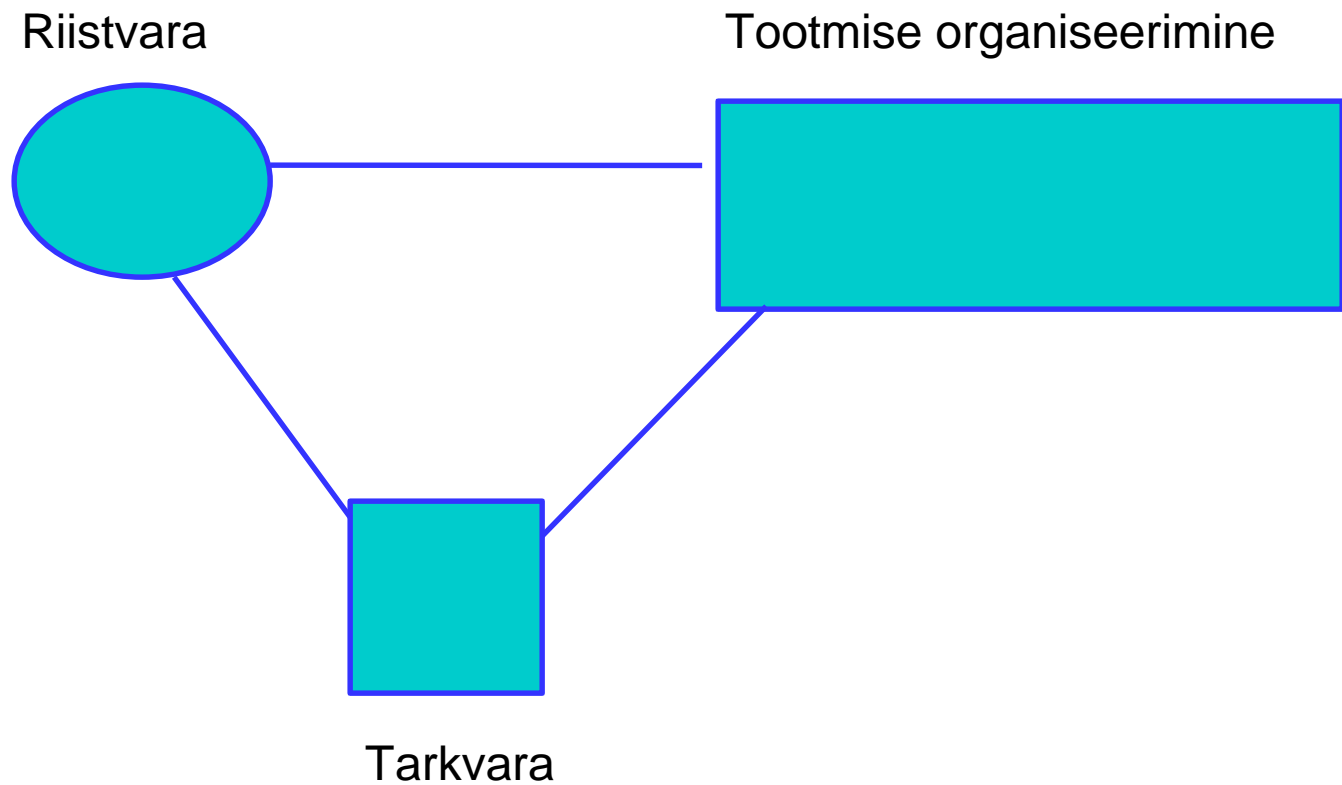
- Versioonikontroll (subversion, git jne)
- Kompileerimissüsteemid (make, automake, Ant, ...)

Testimiseks ja monitoorimiseks:

- Testimise abivahendid ja automaattestide süsteemid (Valgrind, Jtest, ...)
- Automaatmonitoorimise süsteemid

IT süsteemide paradigmade areng

Ajalooline vaade kolmele osale:



Pikaajalised arhitektuurifaasid

1945-1970

1970-1995

1995-2020?

A) **Riistvara** arhitektuurifaasid

Suurarvutid - Mikroarvutid - Võrgusüsteemid

B) **Tarkvaraplatvormide** arhitektuurifaasid:

assembler, puhtad keeled -

teegid, arendusvahendid, komponendid -

komponentide sidumine

C) **Tootmise organiseerimise** arhitektuurifaasid:

suurfirma, avatud - väikefirma, suletud -

vabad komponendid, sidumine, hooldus

Priorities for software development

Three main consumers of time and effort:

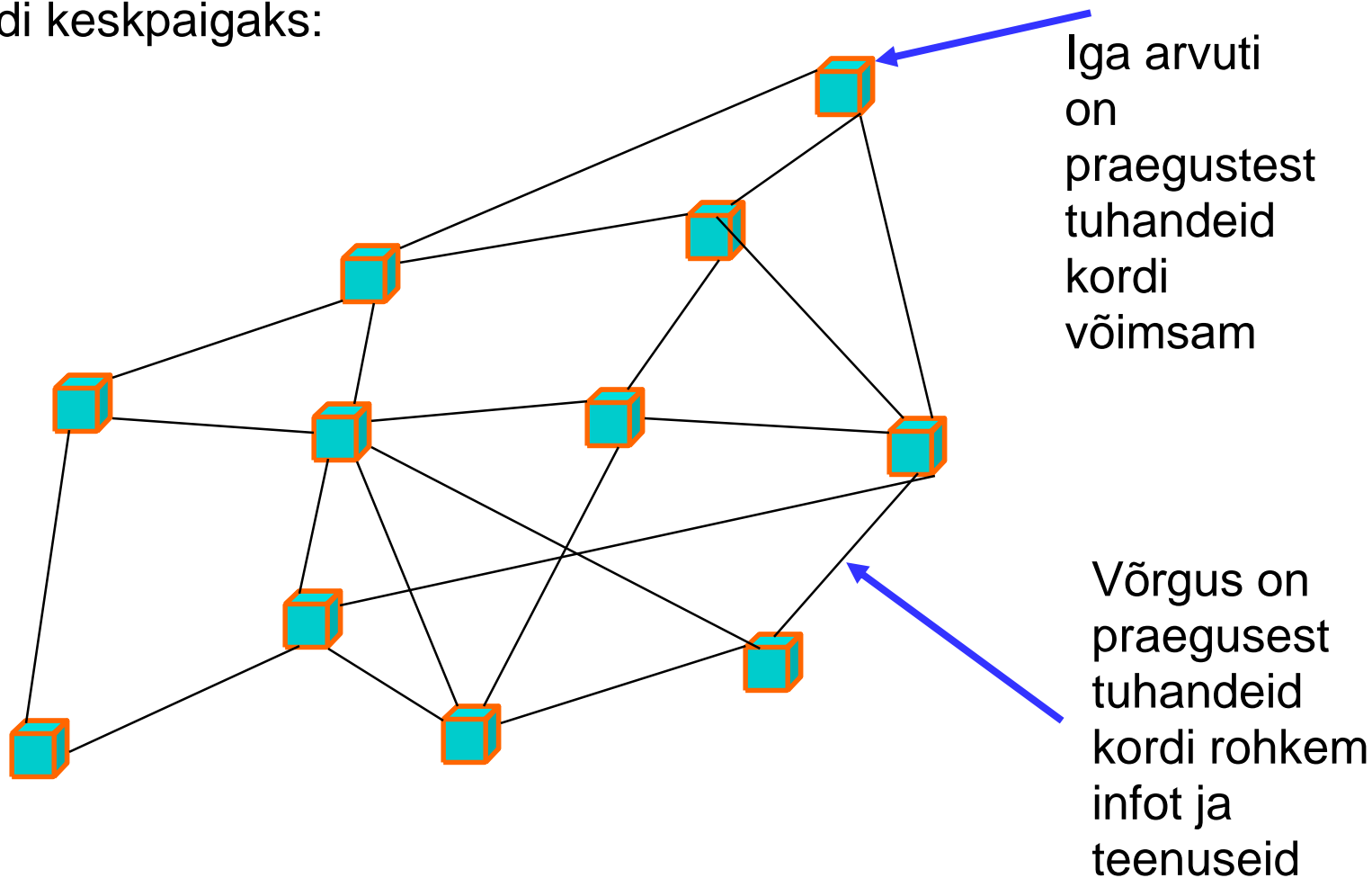
- Understanding the business processes and needs.
- Understanding the exact contents of existing data.
- Writing code.

The second component - understanding existing data - is growing and will keep growing for foreseeable future.

Why?

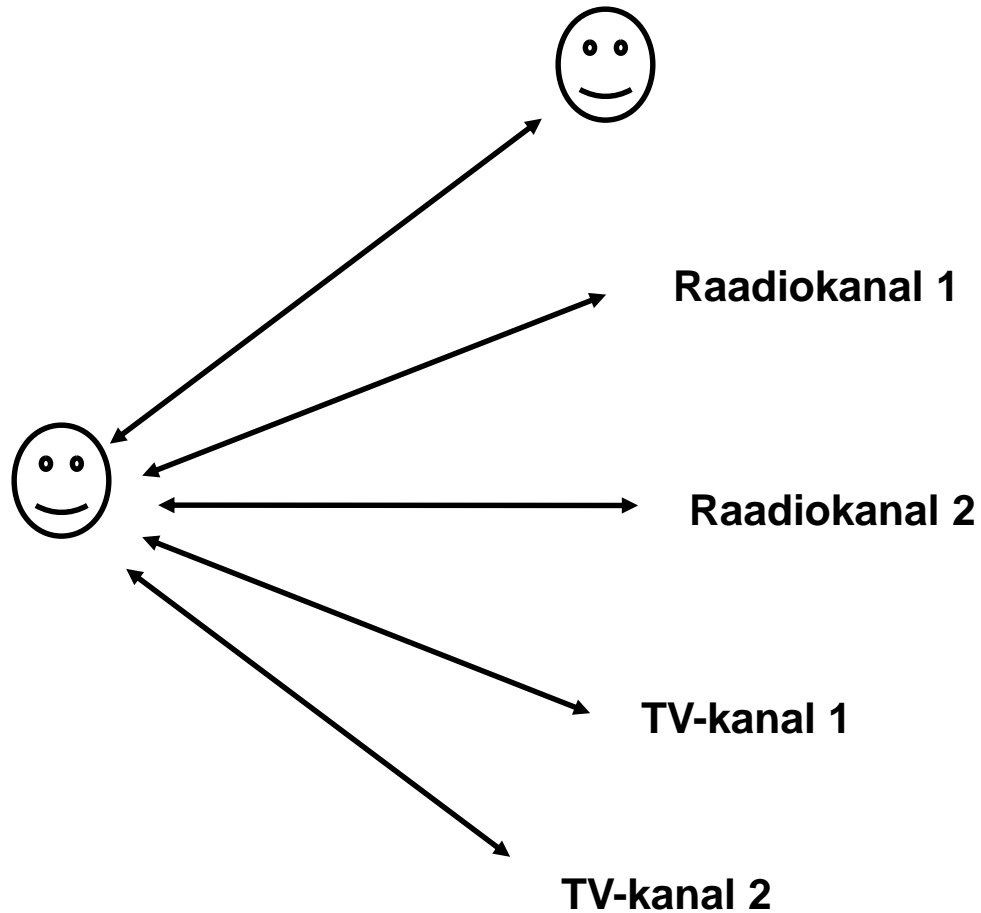
Intelligentne võrk!?

Sajandi keskpaigaks:

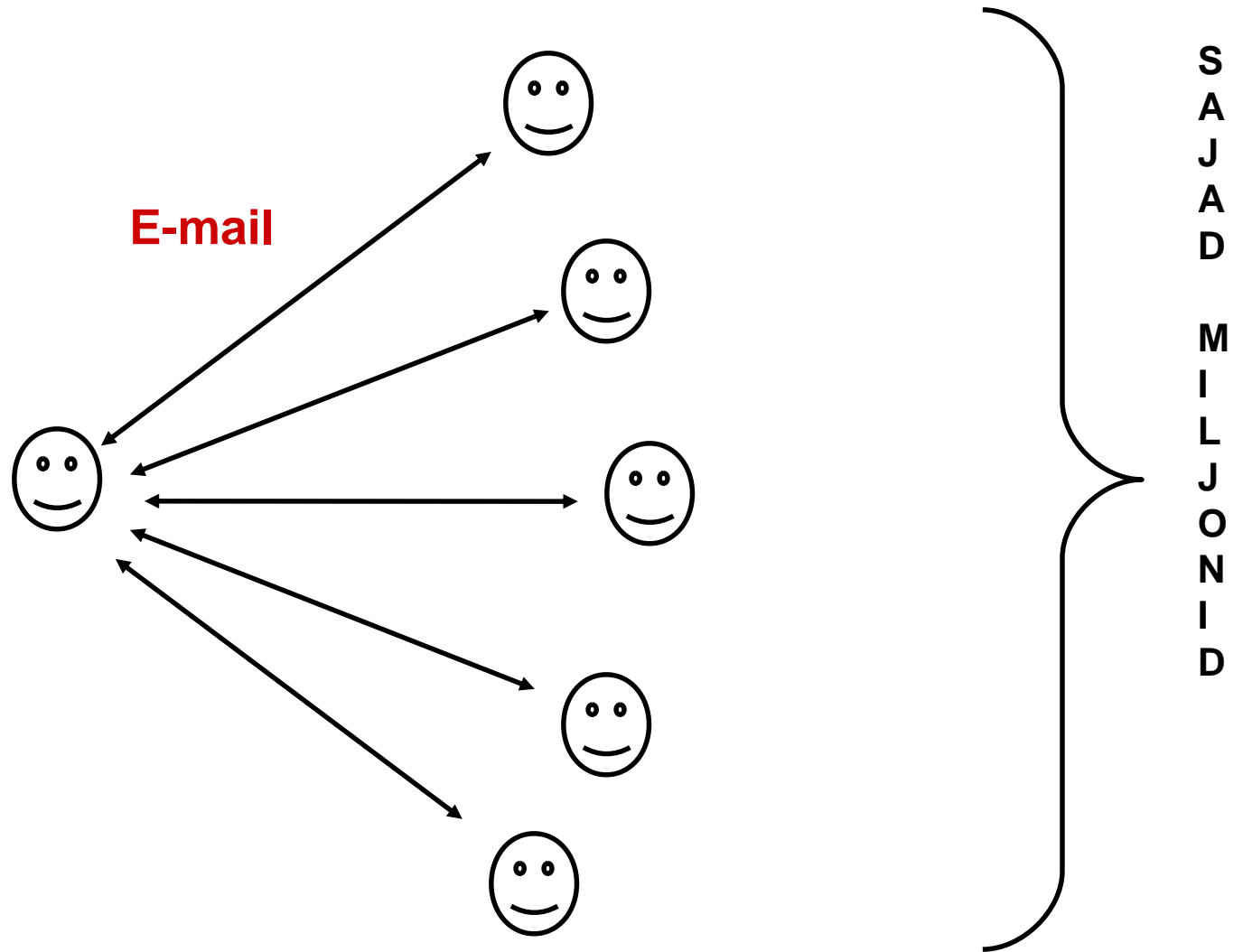


Internet 0:

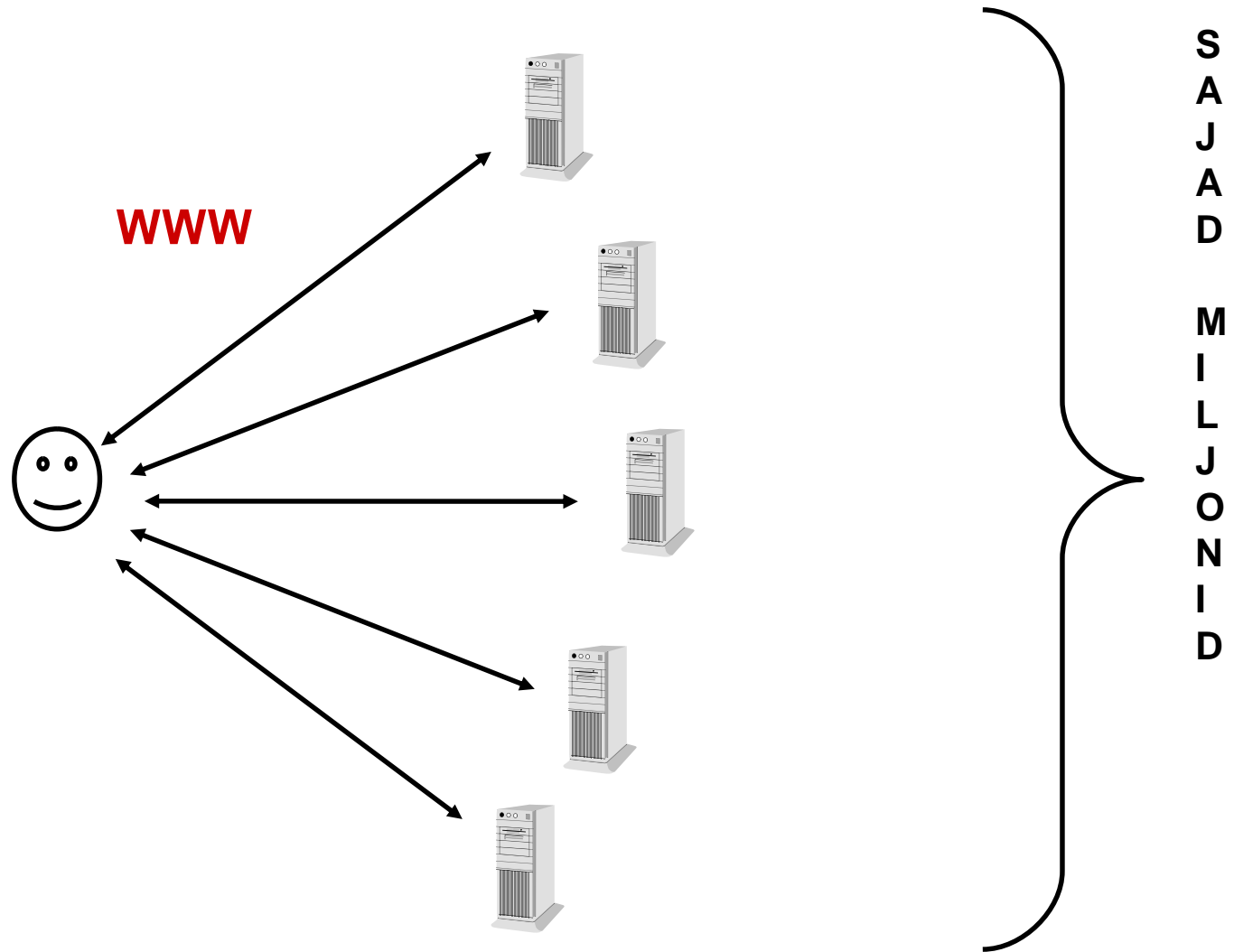
inimene <--> veidi inimesi, raadio, TV



Internet 1: inimene <--> hulga inimesi

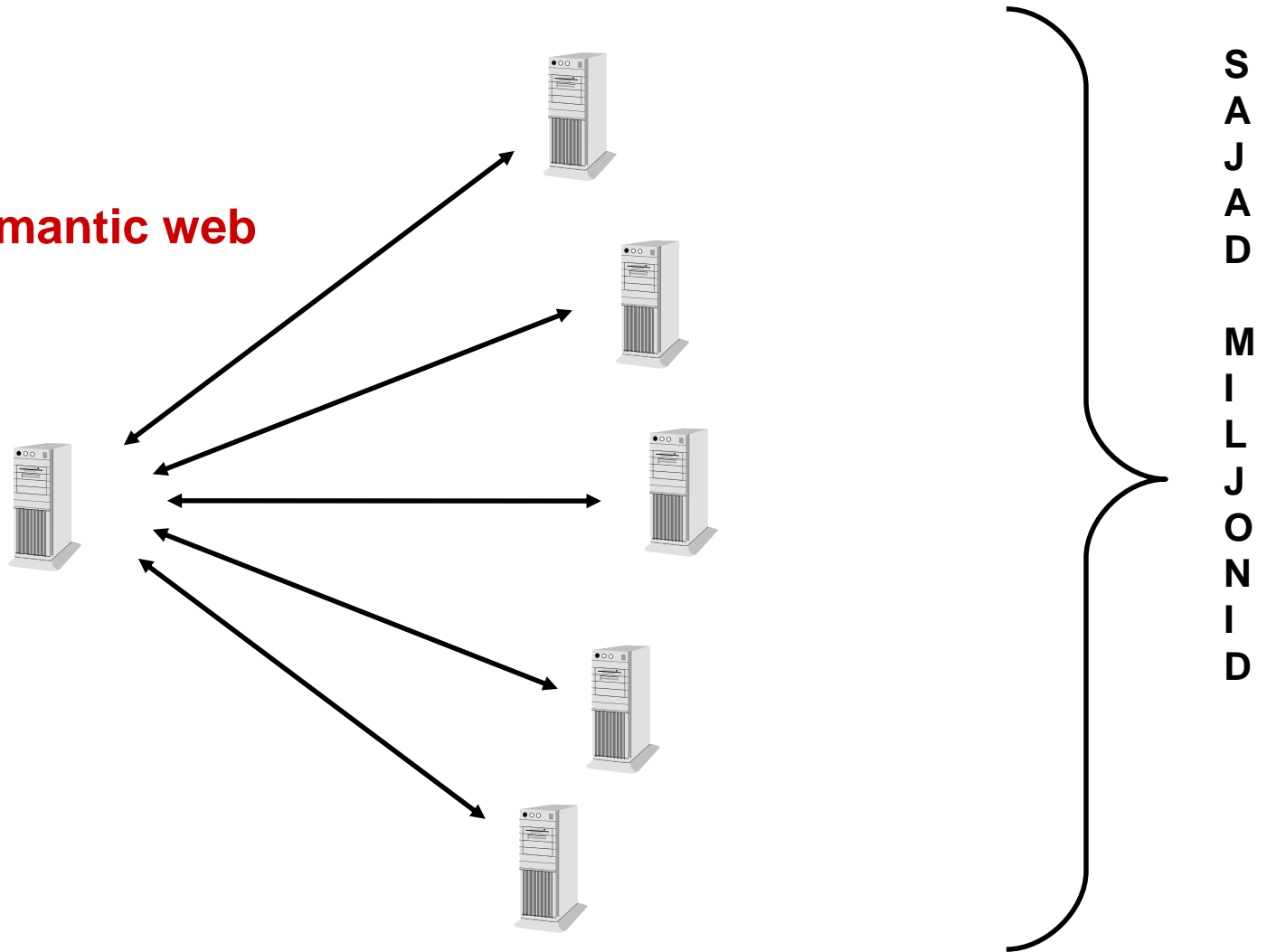


Internet 2: inimene <--> hulga masinaid

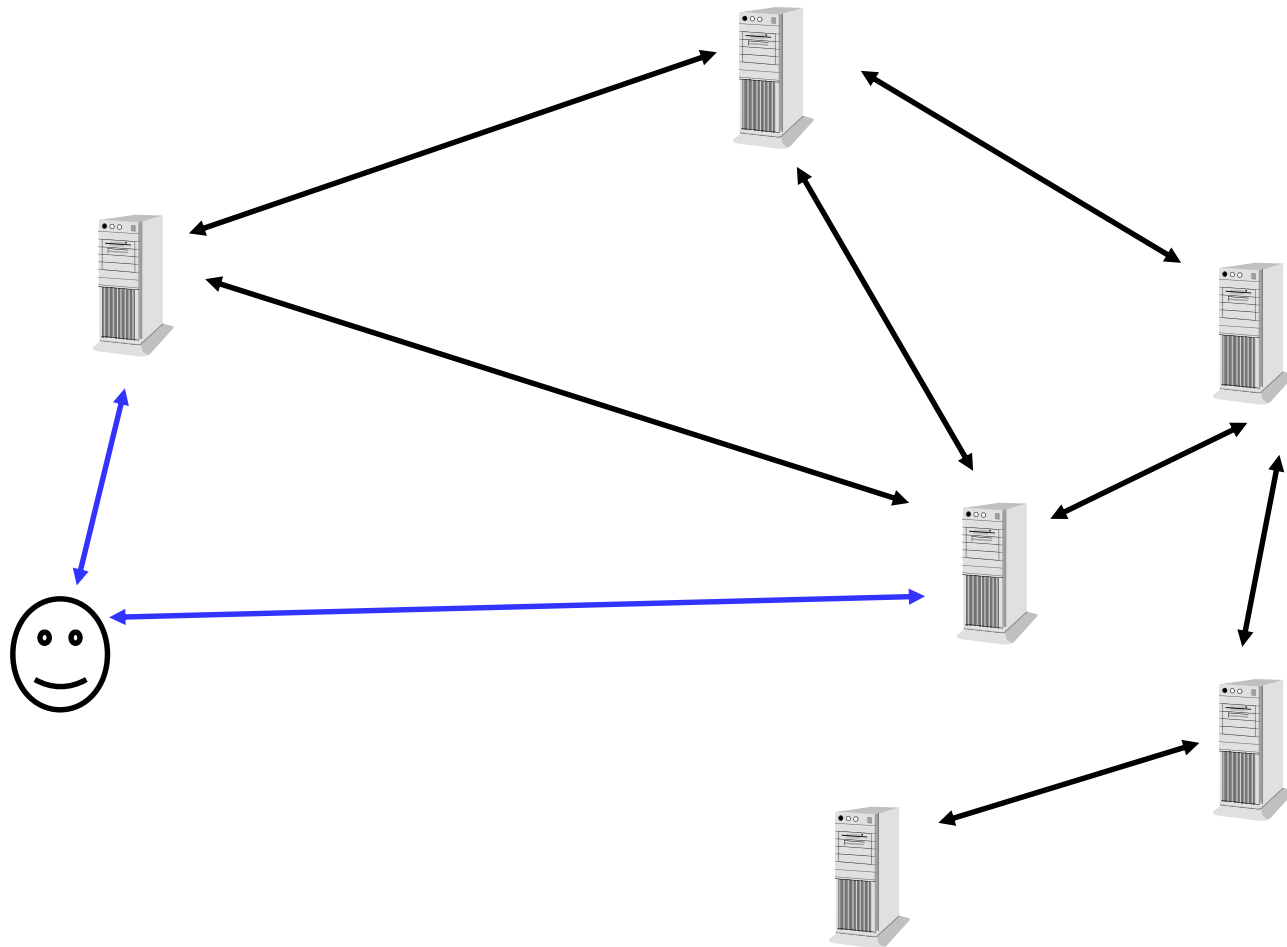


Internet 3: masinad <--> hulga masinaid

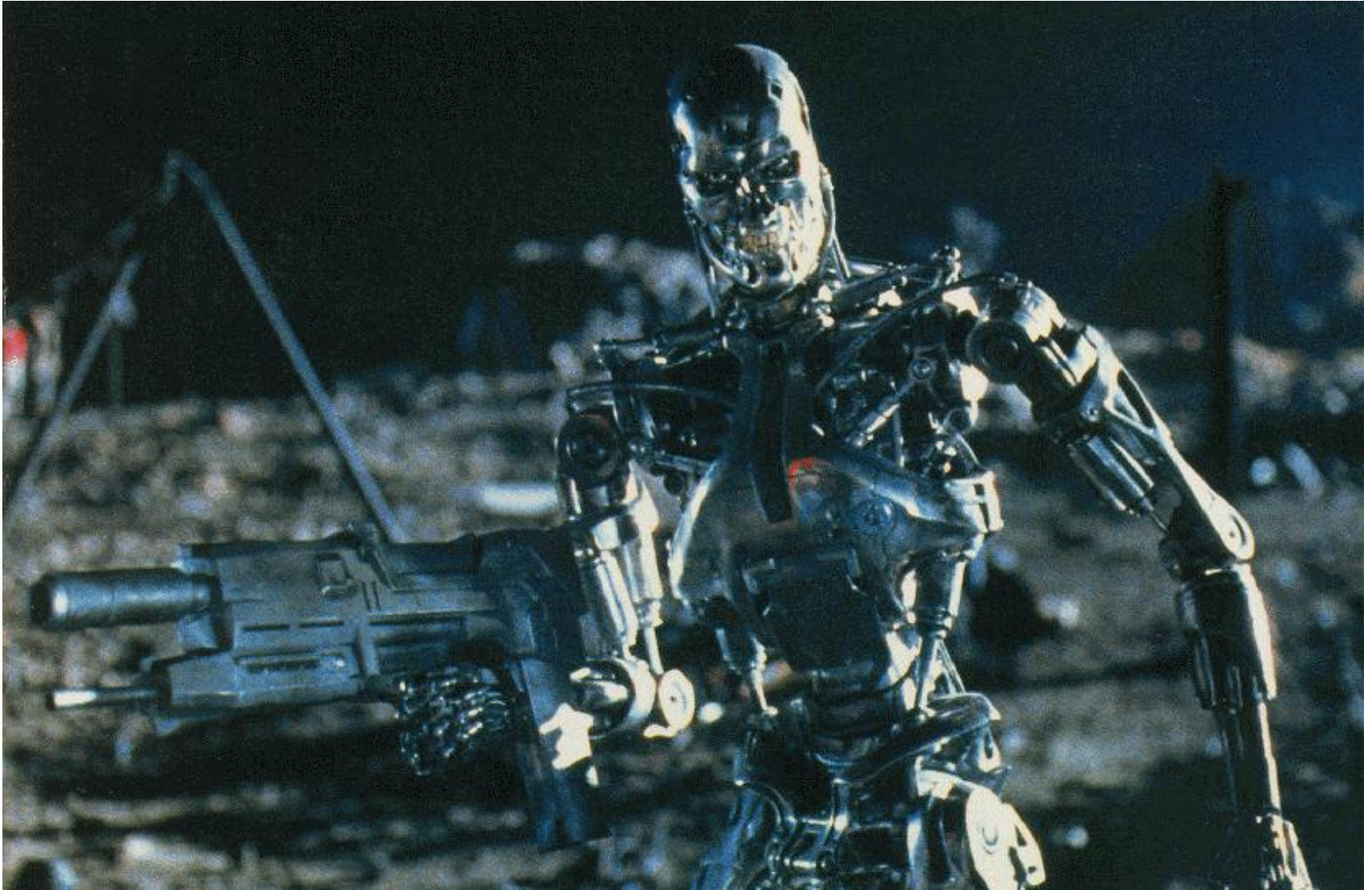
XML + Semantic web



Internet 3: inimene ??

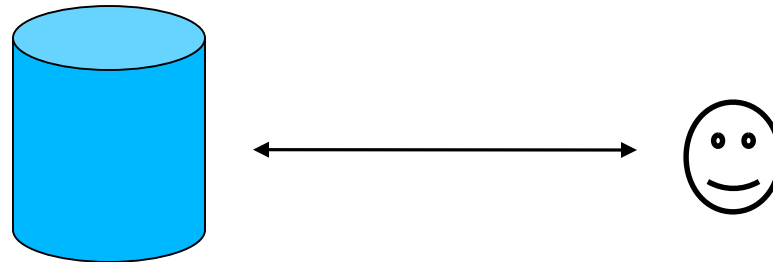


Internet 4: Skynet?



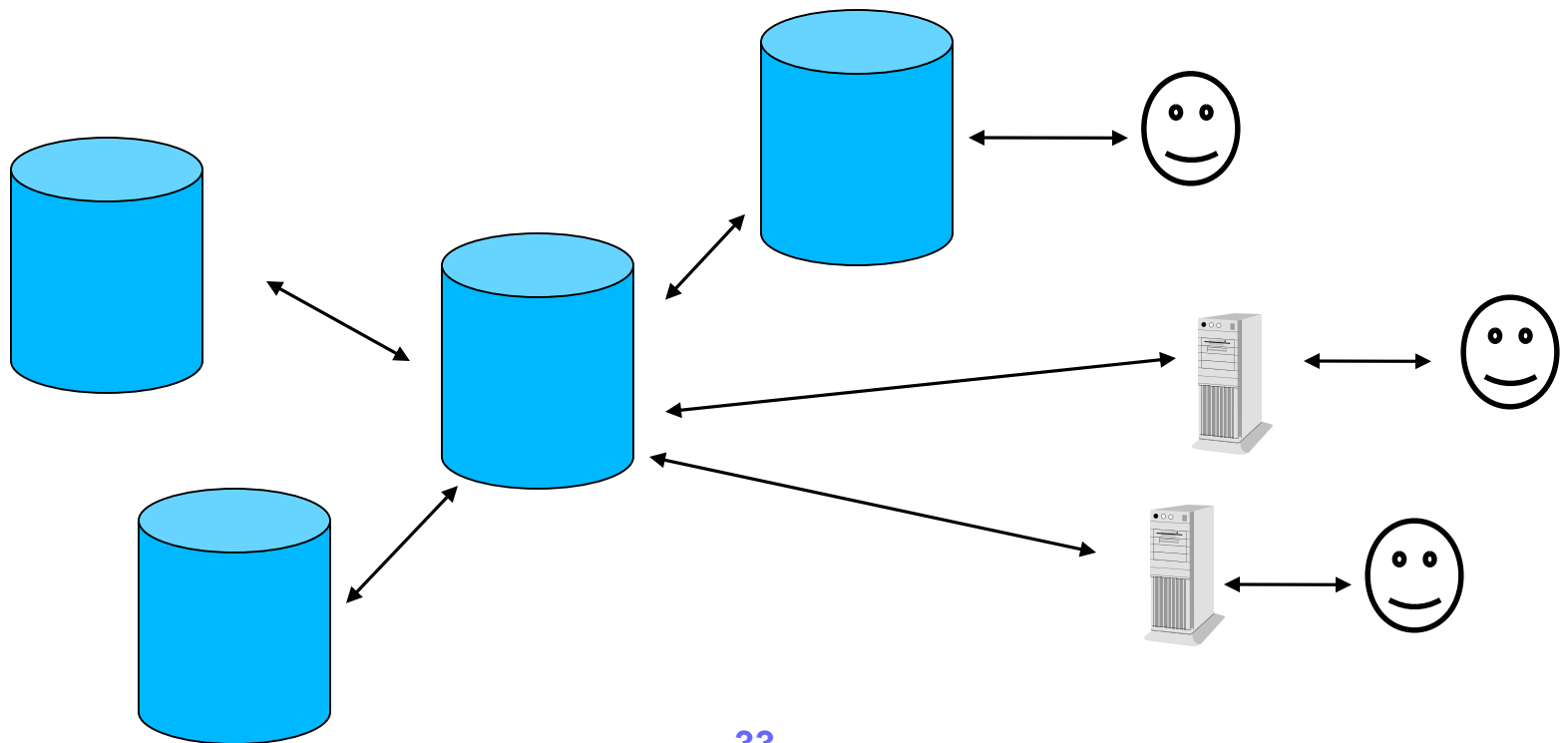
Architecture I

Compact standalone software



Architecture II

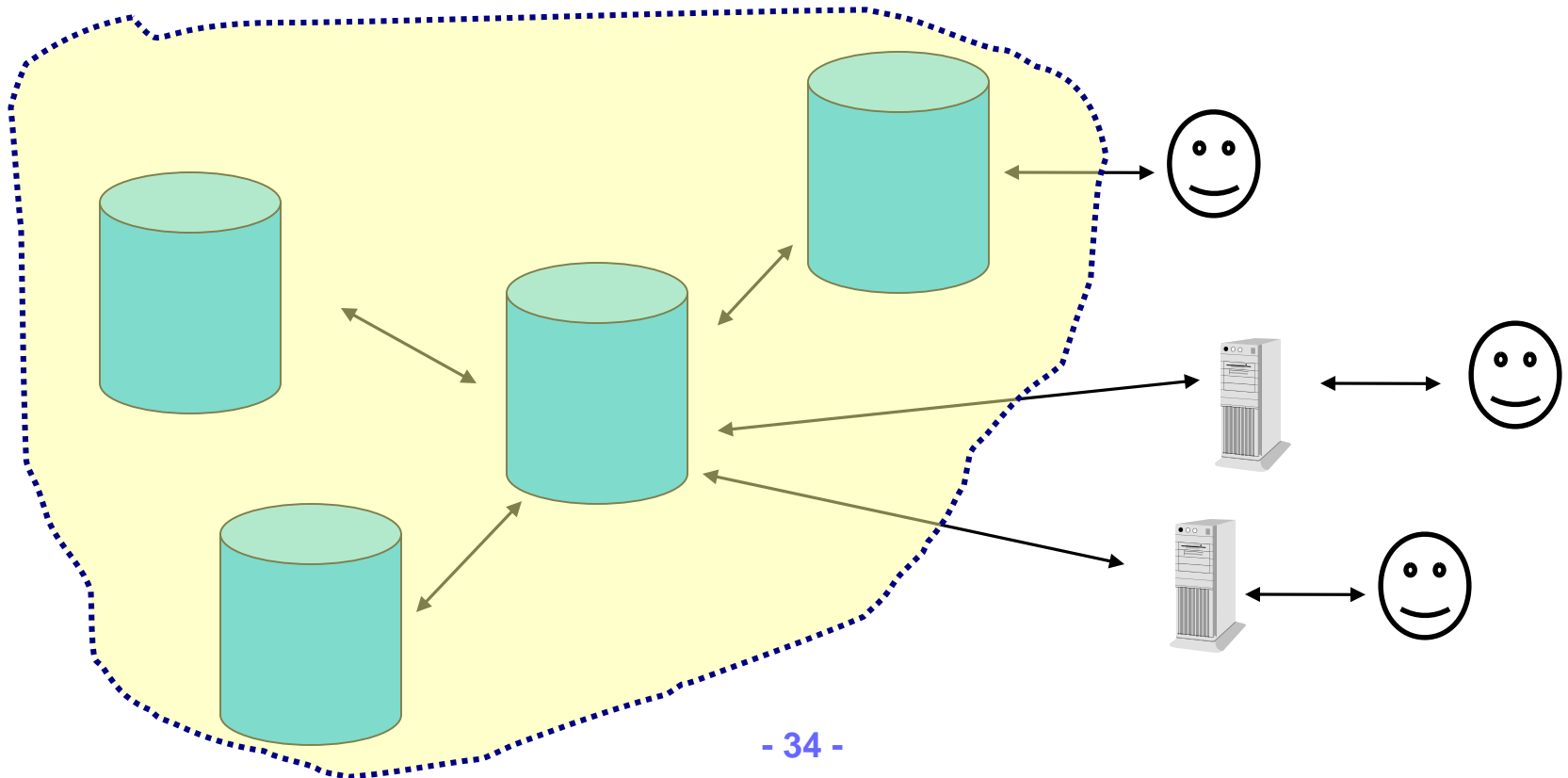
Classical client-server applications



Architecture II : similarities to I

All applications controlled by the developer:

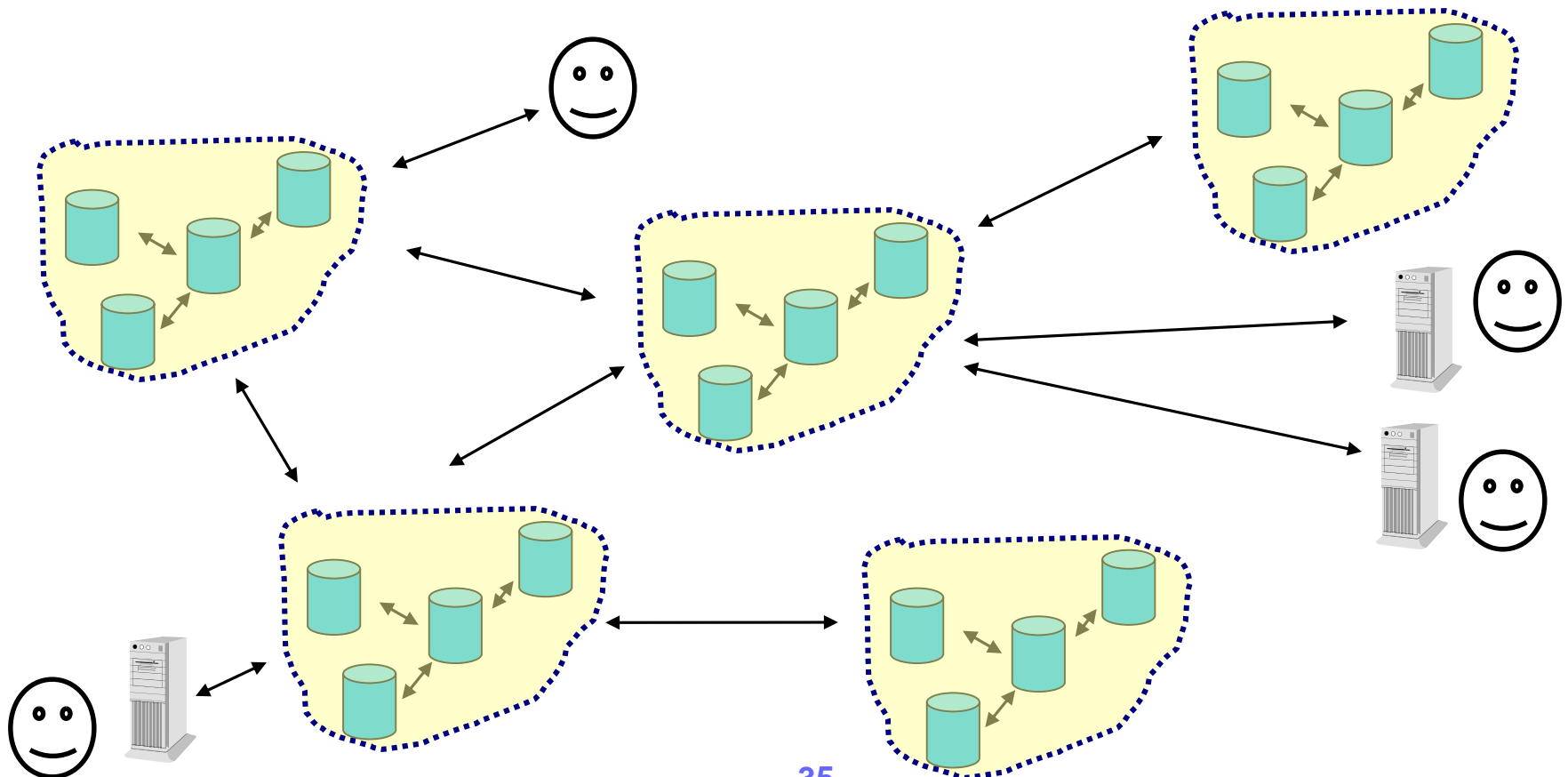
- located in one machine or
- distributed in the machines of the company/client



Architecture III

Applications **not** controlled by the developer:

- you only have access to them
- system functions like a society of applications



Architecture III: some examples

- Apps based on web services
- Global comparison/search systems
- Estonian public sector: xtee complex queries
- Document management between different organisations
- Large financial apps: banks etc
- Sales systems
- Military apps
- ...

Architecture III: kill chain example

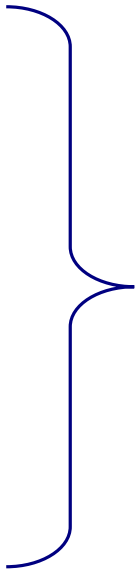
- Context: Yugoslavia, Iraq, Afganistan ...
- Warfighting integration.
- We're basically tasked to close the
- seams in the kill chain.
- The kill chain is a euphemism for the process by which we
 - identify targets
 - find them
 - fix it
 - track it
 - target
 - execute
 - check results



Architecture III: kill chain example

- Context: Yugoslavia, Iraq, Afghanistan ...
- Warfighting integration. We're basically tasked to close the seams in the kill chain. The kill chain is a euphemism for the process by which we

- identify targets
- find them
- fix it
- track it
- target
- execute
- check results

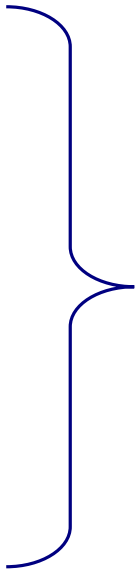


Every link is a separate country with its own information systems

Architecture III: kill chain example

- Context: Yugoslavia, Iraq, Afghanistan ...
- Warfighting integration. We're basically tasked to close the seams in the kill chain. The kill chain is a euphemism for the process by which we

- identify targets
- find them
- fix it
- track it
- target
- execute
- check results



Every link is a separate country with its own information systems

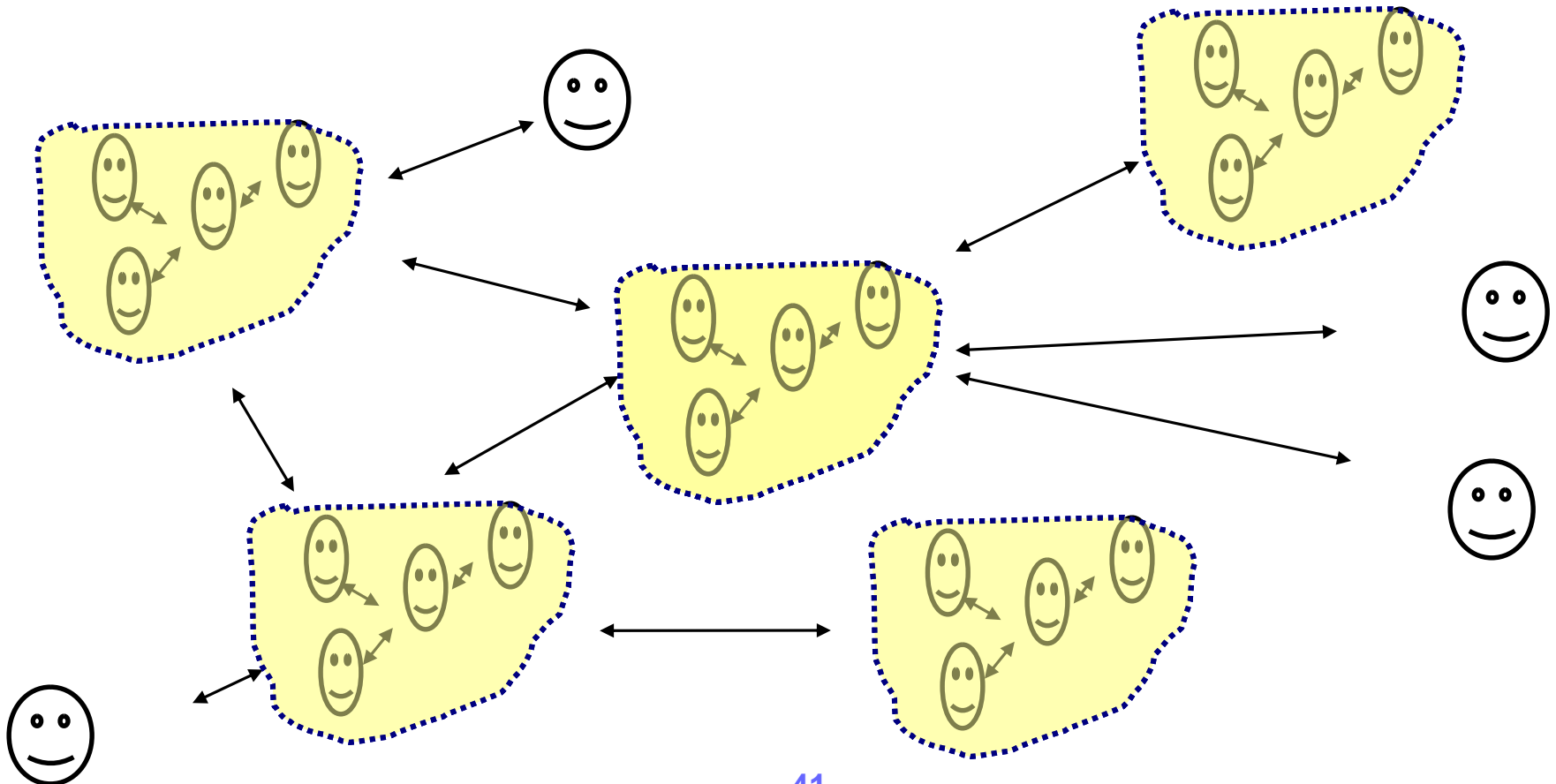
Countries in the chain are often exchanged or replaced!

Main problems for the developer

- **Get access** to data from the foreign system
- **Understand** what does this data exactly mean
- **Convert** foreign data to data structures of our own app and vice versa

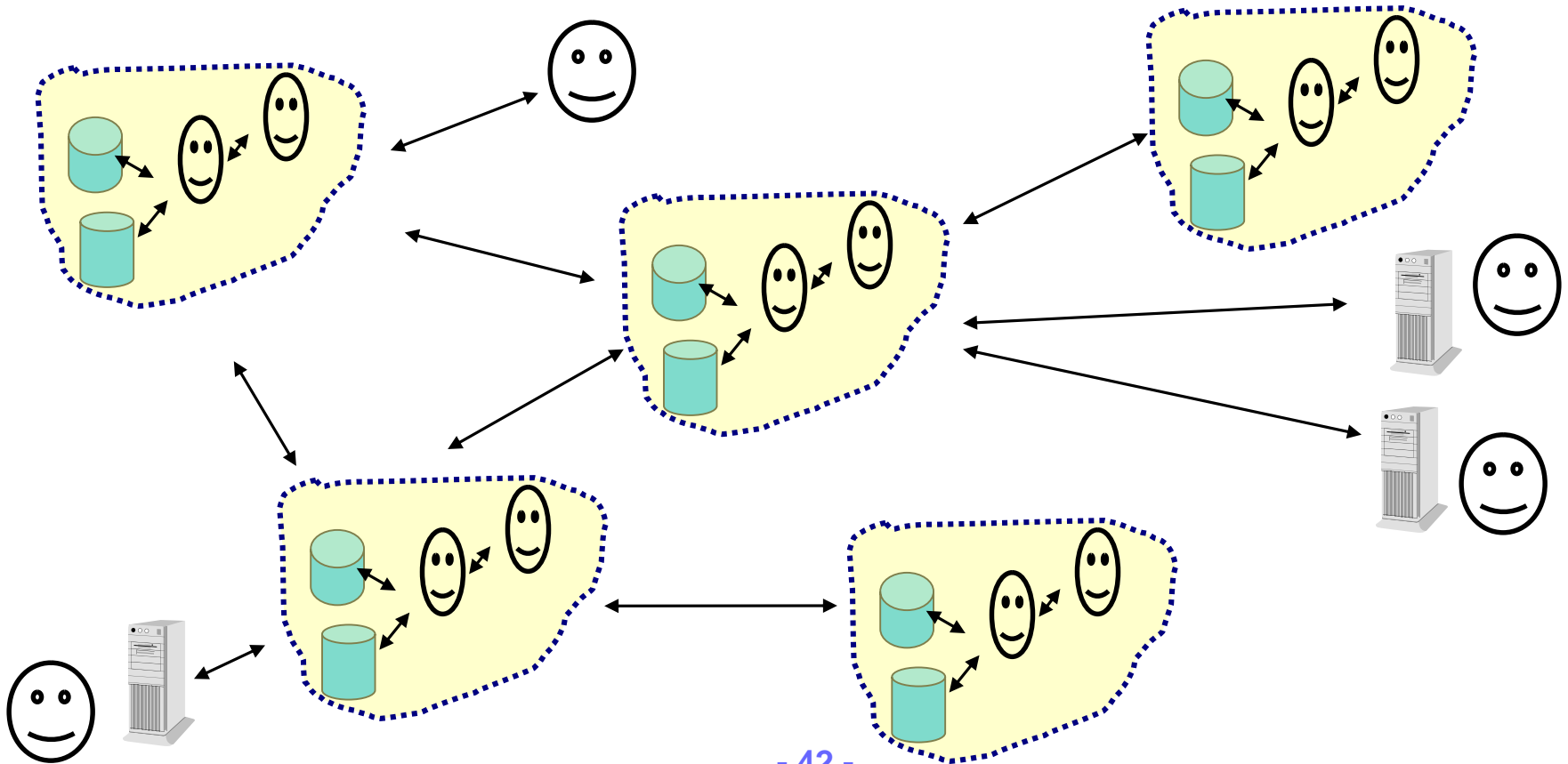
Getting access and understanding?

Social networking. Connected groups of people:



Architecture III revisited

Social groups consist of both people and apps



Philosophy. Society and people.

- We do have AI already:
society is a large animal
with an intelligence of its own.
- **People - society** is similar to
cells - animal



Philosophy. Society and people.

- Society AI benefits from:
 - clever organisation of its brain cells (people)
 - good communication of its brain cells (people)

- Society does not really need very bright people?
- It needs submissive and communicative people?



Philosophy. Notes.

- Both apparently - almost - true:
 - All technologies are turned into guns.
 - All technologies are turned into comm devices.



■ Mis on vabatarkvara:

- tasuta?
- lahtise koodiga?
- edasimüügiõigusega?
- copyright?

■ Ajalugu, eesmärgid, perspektiivid

- mis on olnud?
- mis on tulemas?
- äriidee?
- miks vabavara tehakse?
- kus on vabavara mõistlik kasutada?
- mida vabavara on endaga kaasa toonud?

GNU ideoloogia:

- vabadus: primaarne on tarkvara vabadus, sekundaarne tasuta kättesaadavus (free as in free speech, not as in free beer)
- ausus: ausam on kasutada vabavara kui piraatkopeerida
- teadmiste vabadus: teadmised, tarkvara on loomu poolest vaba - teadmiste ja tarkvara kopeerimine laiendab ühiskonna majanduslikku võimsust, kaotajaid (rumalamaks jääjaid) pole
- raha saab teenida ka tarkvara
 - toetades ja installeerides ja levitades
 - tellitud täiustusi ja modifikatsioone ehitades
 - ehitades sellist vabavara, mida klient soovib
- motivatsioon senistes kogustes tarkvara luua väheneb?
 - vastuargument: programmeerimine on põnev
 - vastuargument: ehk polegi meil nii palju kallilt makstud programmiste vaja?
 - vastuargument: vabanenud programmeerijate kaader saab teha uusi, senisest keerulisemat softi, mille jaoks seni aega ega ressursi ei jätkunud

- Vaatame eraldi GPL, LGPL ja muid vabavara-litsentse